Python.
Модель памяти и ООП

Лекция 5

Преподаватель: *Дмитрий Косицин*

Файлы

Логирование

Модель памяти

Объектно-ориентированное программирование

Тестирование и профилирование

Ответы на вопросы

Файлы

Файлы

```
>>> f = open("path_to_file")  # file object
>>> # some actions
>>> f.close()
```

Если произойдет исключение до метода **close**, файл не будет закрыт.

Используйте менеджер контекстов (context manager):

```
>>> with open("path_to_file") as f:
>>>     # some actions
```

Подробнее о файлах

open(path_to_file, mode, buffering) – открывает файл для работы с ним

- **mode** – способ обращения с файлом: r, w, a, b, t, etc.
- **buffering** – размер буфера

В Python 3 файлы можно читать сразу в кодировке UFT-8 (параметр "encoding")

В Python 2 следует использовать **codecs.open**(...), **codecs.encode**(string, 'utf-8'), **codecs.decode**(string, 'utf-8')

Файлы, по сути, являются последовательностью байтов, а их интерпретация зависит от кодировки.

Работа с файлами

Чтение: методы **read**, **readline**

Запись: метод **write**

Итерирование: **for** line **in** f: ...

Также можно смещаться по файлу (**seek**) и сбрасывать буффер (**flush**).

Важно! Flush не гарантирует запись на диск. Используйте **os.fsync**.

Стандартные потоки ввода-вывода: **sys.stdout** и **sys.stderr**.

Форматы ввода-вывода

В стандартной библиотеке Python поддерживается работа:

- архивами (**zipfile**, **gzip/zlib**)
- **csv**-файлами (самый простой формат таблиц)
- **json** (удобно сохранять словари – json.dumps/json.loads)

Также есть сторонняя библиотека для работы с **yaml** – расширенный и более читаемый **json**.

Для красивого форматирования объектов при печати есть модуль **pprint**.

Логирование

Логирование

```
import sys
import logging
import datetime

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
logger.addHandler(logging.handlers.StreamHandler(sys.stdout))

logger.info("current year's %d", datetime.datetime.today().year)
```

Замечание. В Python 3 Handlers и Filters расположены во вложенных модулях, а содержимое модуля datetime перемещено.

Логирование: подробности

Объекты **Logger** объявлены в модуле **logging**.

Создать новый logger можно вызовом **getLogger**, передав ему некотрое имя.

Вопрос: что будет, если передать имя уже существующего логгера?

Логировать сообщение можно с помощью методов **debug**, **info**, **warning**, **error**, **critical** или общего **log**.

Установить уровень чувствительности (verbosity) можно методов **setLevel**.

Логирование: подробности

Добавить обработчик можно методом **addHandler**, фильтр – **addFilter**.

Реализованные обработчики: **StreamHandler**, **FileHandler**, **RotatingFileHandler**, **SocketHandler**, etc.

Замечание. Все handler'ы и filterer'ы имеют базовые классы - **logging.Handler** и **logging.Filterer**.

Конфигурация логгера может быть сохранена в файле и загружена с помощью **logging.config.dictConfig**.

Модель памяти

...

Объекты

Все сущности в Python являются объектами (наследниками типа ***object***):

- Числа, списки, словари
- Классы и функции
- Код

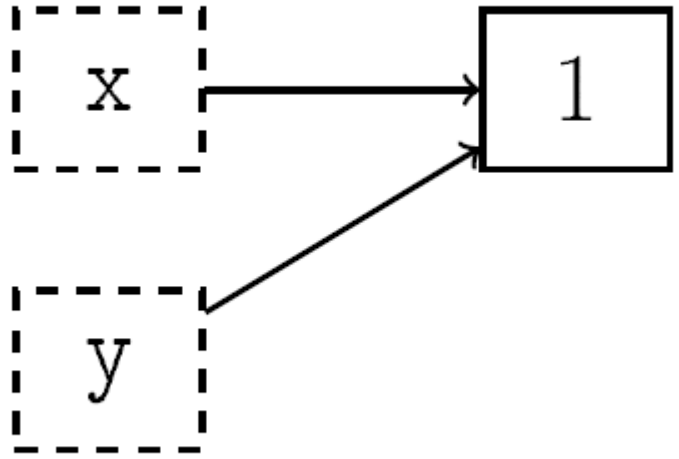
У объектов есть свойства:

- *Идентичность* – адрес в памяти (функция **id**), не изменяется, для сравнения используется **is** и **is not**
- *Тип* определяет допустимые значения и операции над ними, также не изменяется (см. функцию **type**)
- *Значение* может быть *изменяемым* (list) и *неизменяемым* (tuple)

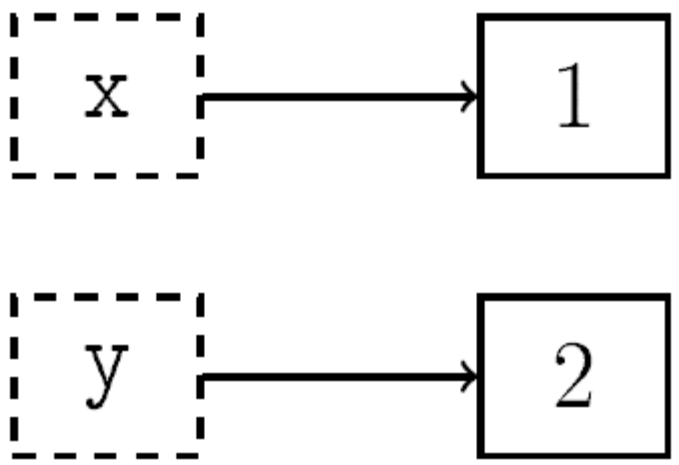
Связь имени и объекта

Каждому имени в коде соответствует объект в памяти:

```
x = 1
y = x
```



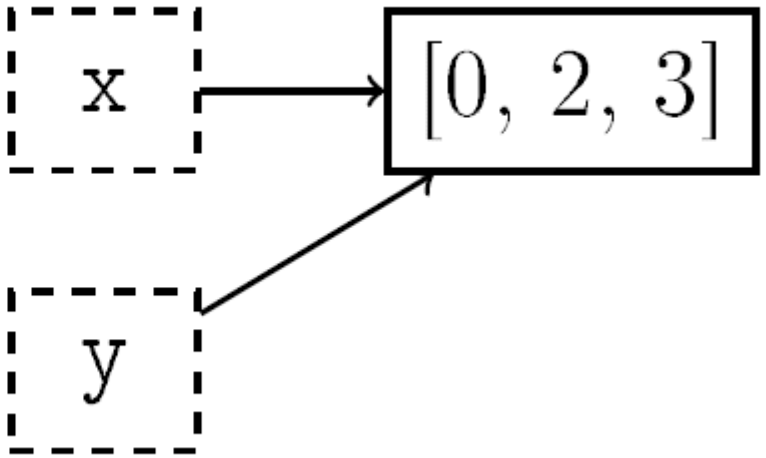

```
x = 1
y = x
y = 2
```



Изменяемые объекты

Значение объектов может меняться:

```
x = [1, 2]
y = x
y += [3]
x[0] = 0
```



Неизменяемые объекты

Значение некоторых объектов, например, кортежей – нет:

```
>>> a = (1, 2)
>>> a[0] = 0
TypeError: 'tuple' object does not support item assignment
```

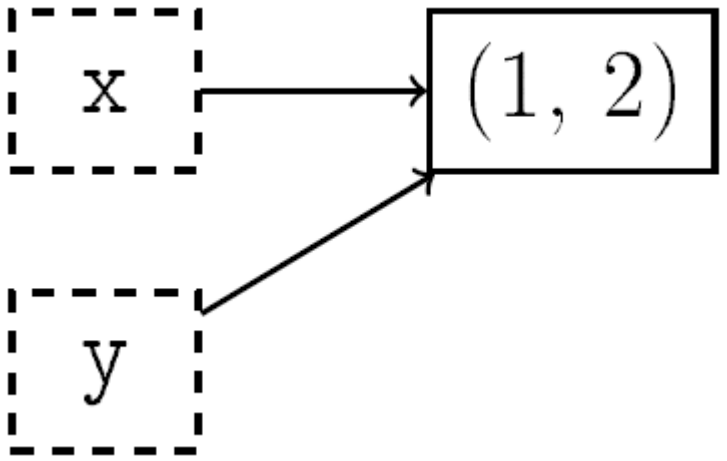
Вопрос: что будет в таком примере?

```
>>> a = ([0], )
>>> a[0] += [1]
```

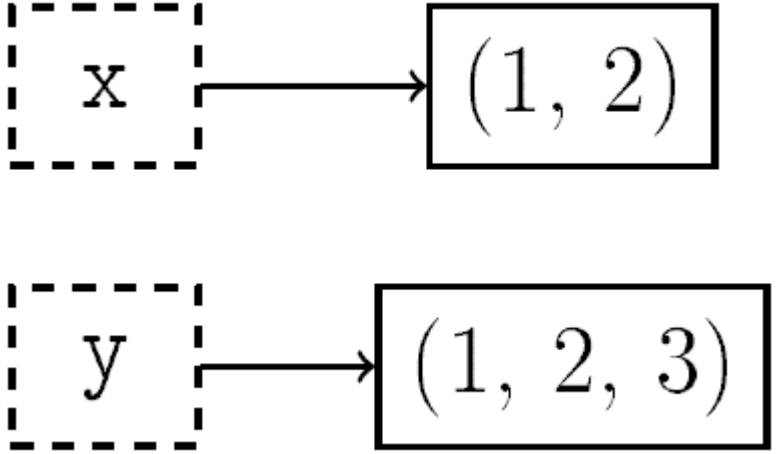
Неизменяемые объекты

При работе с неизменяемыми объектами некоторые операции создадут новый объект:

```
x = (1, 2)
y = x
```



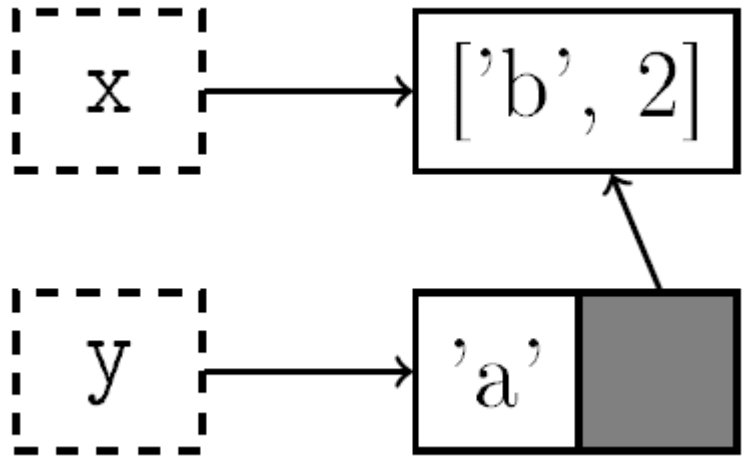

```
x = (1, 2)
y = x
y += (3,)
```



«Контейнеры»

Объекты, например, списки и словари, могут содержать ссылки на другие объекты:

```
x = [1, 2]
y = ('a', x)
x[0] = 'b'
```



Копирование объектов

Копирование объектов

- неизменяемые: при присваивании создается копия
- изменяемые: присваивается ссылка, поэтому нужно использовать либо методы **copy** и **deepcopy** (модуль copy), либо конструктор копирования

Неизменяемые типы

int float complex bool str tuple frozenset

Изменяемые типы

list dict set

Синглтоны

Некоторые объекты поддерживаются в единственном экземпляре:

None True False

А также

- **NotImplemented** (используется в операторах сравнения)
- **Ellipsis** (используется в математических библиотеках)

Замечание. В этом можно убедиться вызвав функцию **id**.

Важно! Объект **None** используется в качестве параметра по умолчанию для обозначения того, что ничего не передано.

Вопрос: а что если **None** является одним из допустимых значений?

Объектно-ориентированное программирование

Классы и объекты

Класс – тип данных, описывает модель некоторой сущности.

Объект – реализация этого класса.

Пример:

int – класс, **42** – объект этого класса (типа **int**)

Пустой класс:

```
>>> class Empty(object):
>>>     pass
```

Методы классов

Функции – вызываемые с помощью скобок объекты.

Методы – функции, которые первым аргументом принимают экземпляр соответствующего класса (обычно именуют его **self**).

```
>>> class Greeter(object):
>>>     def greet(self):
>>>         print "hey, guys!"
```

Атрибуты классов – поля, хранящие некоторые значения (обычно, состояние объекта).

Атрибуты объектов

```
>>> class Greeter(object):
>>>     def set_name(self, name):
>>>         self.name = name
>>>
>>>     def greet(self):
>>>         print "hey, %s!" % self.name
>>>
>>> print(Greeter().greet())
```

Поле классу присвоится run-time после вызова метода *set_name*, поэтому в данном случае произойдет **AttributeError**.

Атрибуты классов

Хорошим правилом будет установка всех атрибутов в конструкторе со значениями по умолчанию или **None**.

```
>>> class Greeter(object):
>>>     DEFAULT_NAME = 'guys'
>>>     def __init__(self, name=None):
>>>         self.name = name or self.__class__.DEFAULT_NAME
```

Здесь DEFAULT_NAME – атрибут класса, к которому можно обращаться:

- по имени класса: Greeter.DEFAULT_NAME
- как к атрибуту класса: self.__class__.DEFAULT_NAME
- как к атрибуту экземпляра класса: self.DEFAULT_NAME

Именование полей класса

Для определения конструктора, переопределения операторов, получения служебной информации в Python используются методы/атрибуты со специальными именами вида __*__ (__*init*__, __*class*__ и т.п.).

Все атрибуты доступны извне класса (являются **public**).

Для обозначения **protected** атрибута используют префикс '_' (underscore), для **private** – '__' (two underscores).

Важно! Доступ к **protected** и **private** атрибутам по-прежнему возможен извне – название служит ***предупреждением***.

Именование полей класса

Преимущества

- Легче отлаживать и проверять код, у IDE больше возможностей
- Легче писать группы классов, связанные друг с другом
- Можно «перехватывать» изменение атрибутов

Замечания

- При желании «защиту» можно обойти
- Многие IDE предупреждают о том, что происходит доступ к **protected** или **private** атрибуту извне класса

Замечание. Обычно private атрибуты используют крайне редко, ведь доступ к ним извне все равно возможен: они доступны как __C_name, где C – имя класса, а name – имя атрибута.

Тестирование и профилирование

Проверка аргументов

assert – statement, позволяющий проверить на истинность некоторое выражение

```
def calculate_binomial_mean(n, p):
    assert n > 0, 'number of experiments must be positive'
    assert 0 <= p <= 1, 'probability must be in [0; 1]'
    return n * p
```

Важно! Скобки **assert** имеют смысл проверки кортежа на пустоту.

Замер времени исполнения

Для замера времени исполнения используйте модуль **timeit**.

```
>>> import timeit
>>> timeit.timeit('"-".join(str(n) for n in
xrange(10000))', number=1000)
2.027786269876367 3
>>> timeit.timeit('"-".join(str(n) for n in range(10000))',
number=1000)
2.2692865730951 44
```

Замечание. Также доступна функция *repeat* (повторять эксперимент несколько раз).

Профилирование

Для профилирования есть модули **cProfile** и **Profile**.

```
>>> import cProfile  # or Profile - pure Python implementation
>>> profiler = cProfile.Profile()
>>> profiler.run_call(calculate_binomial_mean, 10, 0.5)
# another way: run('calculate_binomial_mean(10, 0.5)')
>>> profiler.print_stats()
```

Вызов выведет статистику по времени выполнения функции, в том числе по всем вложенным (если есть).

Замечание. Для просмотра времени выполнения каждой строки используйте, например, **line_profiler**.

Ответы на вопросы

...

Ответы на вопросы

Если передать имя существующего логгера в метод **getLogger**, то будет возвращен уже существующий логгер с таким именем.

При попытке изменить список, который является элементом кортежа с помощью присваивания, произойдет следующее:

- Выполнится сложение – в список добавится новый элемент
- При выполнении присваивания произойдет исключение **TypeError**, что **tuple** не может быть модифицирован

Для корректного изменения списка следует использовать методы **append** и **extend**.

Касательно допустимого значения по умолчанию **None**

Если значение **None** является допустимым и его нельзя использовать в качестве аргумента по умолчанию, то создают объект (например, глобальный) типа **object** и проверяют, является ли переданный аргумент идентичным этому объекту.

```
>>> not_set = object()
>>> def f(x=not_set):
>>>     is_x_set = (x is not not_set)
```